LEARN C++
Skill up with our free tutorials

1.9 — Introduction to literals and operators

ALEX[1] APRIL 8, 2024

Literals

Consider the following two statements:

```
std::cout << "Hello world!";
int x { 5 };
```

What are “”Hello world!”” and ‘5’? They are literals. A **literal** (also known as a **literal constant**) is a fixed value that has been inserted directly into the source code.

Literals and variables both have a value (and a type). Unlike a variable (whose value can be set and changed through initialization and assignment respectively), the value of a literal is fixed (`5` is always `5`). This is why literals are called constants.

To further highlight the difference between literals and variables, let's examine this short program:

```
1   #include <iostream>
2
3   int main()
4   {
5       std::cout << 5 << '\n'; // print the value of a literal
6
7       int x { 5 };
8       std::cout << x << '\n'; // print the value of a variable
9       return 0;
10  }
```

On line 5, we're printing the value `5` to the console. When the compiler compiles this, it will generate code that causes `std::cout` to print the value `5`. This value `5` is compiled into the executable and can be used directly.

On line 7, we're creating a variable named `x`, and initializing it with value `5`. The compiler will generate code that copies the literal value `5` into whatever memory location is given to `x`. On line 8, when we print `x`, the compiler will generate code that causes `std::cout` to print the value at the memory location of `x` (which has value `5`).

Thus, both output statements do the same thing (print the value 5). But in the case of the literal, the value `5` can be printed directly. In the case of the variable, the value `5` must be fetched from the memory the variable represents.

This also explains why a literal is constant while a variable can be changed. A literal's value is placed directly in the executable, and the executable itself can't be changed after it is created. A variable's value is placed in memory, and the value of memory can be changed while the executable is running.

Key insight

Literals are values that are inserted directly into the source code. These values usually appear directly in the executable code (unless they are optimized out).

Objects and variables represent memory locations that hold values. These values can be fetched on demand.

Related content

We talk more about literals in lesson 5.2 -- Literals (https://www.learncpp.com/cpp-tutorial/literals/)[2].

Operators

In mathematics, an **operation** is a process involving zero or more input values (called **operands**) that produces a new value (called an *output value*). The specific operation to be performed is denoted by a symbol called an **operator**.

For example, as children we all learn that *2 + 3* equals *5*. In this case, the literals *2* and *3* are the operands, and the symbol *+* is the operator that tells us to apply mathematical addition on the operands to produce the new value *5*.

In C++, operations work as you'd expect. For example:

```cpp
#include <iostream>

int main()
{
    std::cout << 1 + 2 << '\n';

    return 0;
}
```

In this program, the literals `1` and `2` are operands to the plus (`+`) operator, which produces the output value `3`. This output value is then printed to the console. In C++, the output value of an operation is often called a **return value**.

You are likely already quite familiar with standard arithmetic operators from common usage in mathematics, including addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`). In C++, assignment (`=`) is an operator as well, as are insertion (`<<`), extraction (`>>`), and equality (`==`). While most operators have symbols for names (e.g. `+`, or `==`), there are also a number of operators that are keywords (e.g. `new`, `delete`, and `throw`).

Author's note

For reasons that will become clear when we discuss operators in more detail, for operators that are symbols, it is common to append the operator's symbol to the word *operator*.

For example, the plus operator would be written `operator+`, and the extraction operator would be written `operator>>`.

The number of operands that an operator takes as input is called the operator's **arity**. Few people know what this word means, so don't drop it in a conversation and expect anybody to have any idea what you're talking about. Operators in C++ come in four different arities:

Unary operators act on one operand. An example of a unary operator is the `-` operator. For example, given `-5`, `operator-` takes literal operand `5` and flips its sign to produce new output value `-5`.

Binary operators act on two operands (often called *left* and *right*, as the left operand appears on the left side of the operator, and the right operand appears on the right side of the operator). An example of a binary operator is the `+` operator. For example, given `3 + 4`, `operator+` takes the left operand `3` and the right operand `4` and applies mathematical addition to produce new output value `7`. The insertion (`<<`) and extraction (`>>`) operators are binary operators, taking `std::cout` or `std::cin` on the left side, and the value to output or variable to input to on the right side.

Ternary operators act on three operands. There is only one of these in C++ (the conditional operator), which we'll cover later.

Nullary operators act on zero operands. There is also only one of these in C++ (the throw operator), which we'll also cover later.

Note that some operators have more than one meaning depending on how they are used. For example, `operator-` has two contexts. It can be used in unary form to invert a number's sign (e.g. to convert `5` to `-5`, or vice versa), or it can be used in binary form to do subtraction (e.g. `4 - 3`).

Chaining operators

Operators can be chained together such that the output of one operator can be used as the input for another operator. For example, given the following: `2 * 3 + 4`, the multiplication operator goes first, and converts left operand `2` and right operand `3` into return value `6` (which becomes the left operand for the plus operator). Next, the plus operator executes, and converts left operand `6` and right operand `4` into new value `10`.

We'll talk more about the order in which operators execute when we do a deep dive into the topic of operators. For now, it's enough to know that the arithmetic operators execute in the same order as they do in standard mathematics: Parenthesis first, then Exponents, then Multiplication & Division, then Addition & Subtraction. This ordering is sometimes abbreviated *PEMDAS*, or expanded to the mnemonic "Please Excuse My Dear Aunt Sally".

Author’s note

In some countries, PEMDAS is taught as PEDMAS, BEDMAS, BODMAS, or BIDMAS instead.

Return values and side effects

Most operators in C++ just use their operands to calculate a return value. For example, `-5` produces return value `-5` and `2 + 3` produces return value `5`. There are a few operators that do not produce return values (such as `delete` and `throw`). We’ll cover what these do later.

Some operators have additional behaviors. An operator (or function) that has some observable effect beyond producing a return value is said to have a **side effect**. For example, when `x = 5` is evaluated, the assignment operator has the side effect of assigning the value `5` to variable `x`. The changed value of `x` is observable (e.g. by printing the value of `x`) even after the operator has finished executing. `std::cout << 5` has the side effect of printing `5` to the console. We can observe the fact that `5` has been printed to the console even after `std::cout << 5` has finished executing.

Operators with side effects are usually called for the behavior of the side effect rather than for the return value (if any) those operators produce.

Nomenclature

In common language, the term “side effect” is typically used to mean a secondary (often negative or unexpected) result of some other thing happening (such as taking medicine). For example, a common side effect of taking oral antibiotics is diarrhea. As such, we often think of side effects as things we want to avoid, or things that are incidental to the primary goal.

In C++, the term “side effect” has a different meaning: it is an observable effect of an operator or function beyond producing a return value.

Since assignment has the observable effect of changing the value of an object, this is considered a side effect. We use certain operators (e.g. the assignment operator) primarily for their side effects. In such cases, the side effect is both beneficial and predictable (and it is the return value that is often incidental).

For advanced readers

For the operators that we call primarily for their return values (e.g. `operator+` or `operator*`), it’s usually obvious what their return values will be (e.g. the sum or product of the operands).

For the operators we call primarily for their side effects (e.g. `operator=` or `operator<<`), it’s not always obvious what return values they produce (if any). For example, what return value would you expect `x = 5` to have?

Both `operator=` and `operator<<` (when used to output values to the console) return their left operand. Thus, `x = 5` returns `x`, and `std::cout << 5` returns `std::cout`. This is done so that these operators can be chained.

For example, `x = y = 5` evaluates as `x = (y = 5)`. First `y = 5` assigns `5` to `y`. This operation then returns `y`, which can then be assigned to `x`.

`std::cout << "Hello " << "world"` evaluates as `(std::cout << "Hello ") << "world!"`. This first prints `"Hello "` to the console. This operation returns `std::cout`, which can then be used to print `"world!"` to the console as well.

We talk more about the order in which operators evaluate in lesson 6.1 -- Operator precedence and associativity (https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/)[3].

Quiz time

Question #1

For each of the following, indicate what output they produce:

a)

```
std::cout << 3 + 4 << '\n';
```

Hide Solution (javascript:void(0))[4]

7

b)

```
std::cout << 3 + 4 - 5 << '\n';
```

Hide Solution (javascript:void(0))[4]

2

c)

```
std::cout << 2 + 3 * 4 << '\n';
```

Hide Solution (javascript:void(0))[4]

14. Multiplication goes before addition, so 3 * 4 goes first, producing the result 12. 2 + 12 is 14.

d) Extra credit:

```cpp
int x { 2 };
std::cout << (x = 5) << '\n';
```

Hide Solution (javascript:void(0))[4]

5. `x = 5` assigns the value `5` to `x`, and then returns `x`. The value of `x` (now `5`) is then printed to the console.

Next lesson
1.10 Introduction to expressions

5

Back to table of contents

6

Previous lesson
1.8 Whitespace and basic formatting

7

8

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[11] are connected to your provided email address.

213 COMMENTS Newest

Charlie
April 8, 2024 7:37 am

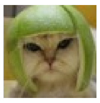
The "For Advanced Readers"-section about operators like = and << and their return values was really insightful!

0 Reply

Likss
April 6, 2024 12:43 pm

I am unable to understand how int x = 5 has an side effect . Main job of = is to assign the value to the variable . It does that . So how is it a side effect . May be i am being dumb . Kindly explain in details

0 Reply



Alex Author
Reply to Likss [12] April 8, 2024 8:47 am

From the article: "In C++, the term “side effect” has a different meaning: it is an observable effect of an operator or function beyond producing a return value."

Since assignment has the observable effect of changing the value of a variable, this is considered a side effect.

0 Reply

Likss
Reply to Alex [13] April 9, 2024 12:57 am

Thanks

0 Reply

tausiqsama
Reply to Likss [12] April 7, 2024 10:57 pm

Do `int i = 7.5;` and then do `int i {7.5};` and tell us the answer

0 Reply

sajLMAO
Reply to tausiqsama [14] April 12, 2024 6:50 pm

maybe use double instead of int

0 Reply

Likss

Reply to tausiqsama [14] April 9, 2024 12:57 am

I do not understand what you are saying

0 Reply

Aman Jaiswal

March 23, 2024 7:48 am

"For example, x = y = 5 evaluates as x = (y = 5). First y = 5 assigns 5 to y. This operation then returns y, which can then be assigned to x."

In the above, The operation doesn't return the variable y itself but rather the value that is assigned to y.
So a better way to say would be " First, 5 is assigned to y. This operation then returns the value 5. The returned value 5 is then assigned to x "

0 Reply

Alex Author

Reply to Aman Jaiswal [15] March 24, 2024 1:45 pm

We can prove this is untrue experimentally:

```
int main()
{

    int y{};
    (y = 5) = 6;
    std::cout << y << '\n';

    return 0;
}
```

If `(y = 5)` returns a value or a temporary, then assigning `6` to that result should not affect `y`, and the subsequent print statement should print `5`.

However, the subsequent statement prints `6`, showing that `y = 5` actually returned `y`, which was then assigned the value `6`.

1 Reply

Aman Jaiswal

Reply to Alex [16] March 26, 2024 9:13 am

gotcha.
thank you for clarifying in detail:)

1 Reply

Sangeet
March 3, 2024 1:55 am

woah, the return values of the assignment operator and the insertion operator is a cool thing to know.

3 Reply

naul1
Reply to Sangeet [17] March 4, 2024 10:04 am

agreed

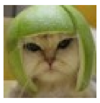
Last edited 1 month ago by naul1

0 Reply

IceFloe
February 18, 2024 4:47 pm

does a unary operator always have priority over other operators? in theory, the expression x = -5*7+-2/2 , first, the values 5 and 2 are assigned a minus sign, and then multiplication and addition take place?

2 Reply



Alex Author
Reply to IceFloe [18] February 18, 2024 9:28 pm

We have a table of operator priority in lesson https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/

But yes, unary plus and unary minus take precedence over the binary arithmetic operators.

2 Reply

Piyush Yadav
February 5, 2024 10:32 pm

just out of curiosity i tried reading a bit on insertion operator and the return type
I have knowledge of C++ fundamentals, will i be able to understand this deep after understanding all the content on this website :/ like what basic_ostream<char> is

```cpp
#include <iostream>

std::basic_ostream<char> &test()
{
    return (std::cout << 5);
}

int main()
{
    test() << 6;
    return 0;
}

// output 56
```

0 Reply

Alex Author

Reply to Piyush Yadav [19] February 6, 2024 1:08 pm

Yes. FWIW, you can use `std::ostream` instead of `std::basic_ostream<char>`.

0 Reply

Elias
February 4, 2024 4:59 pm

Thank you so much for a truly valuable content. I am more a reader than a watcher, and this is perfect in both pace and detail. You're a legend Alex. I hope to give back to you one day - keep the donation button up.

5 Reply

Ajit Mali
January 25, 2024 2:30 am

While most operators have symbols for names (e.g. +, or ==),

Can any one explain what ut says

0 Reply

Alex Author

Reply to Ajit Mali [20] January 25, 2024 9:45 pm

A symbol is a non-letter/non-number, like +, =, ! etc...

The `+` operator has the `+` symbol for it's name. As opposed to something like `sizeof`, which is an operator that has a non-symbol name.

1 Reply

Abhey
January 18, 2024 5:43 am

why do we have to use bracket for the last question and why just writing `x = 5` is not working and giving me error `no match for 'operator='`

Last edited 2 months ago by Abhey

0 Reply

CDEE
Reply to Abhey 21 March 3, 2024 2:15 am

```
#include <iostream>

int main()
{
int x{ 2 };
std::cout << (x = 5) << '\n';
}
```
Try this.

0 Reply

Sangeet
Reply to Abhey 21 March 3, 2024 2:02 am

Hi Abhey, I actually found the answer to your question in a comment below.

https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-591508

1 Reply

Long
Reply to Abhey 21 January 23, 2024 7:18 pm

it works normal for me

0 Reply

DDDD
Reply to Abhey[21] January 21, 2024 6:48 pm

If you use Visual Studio, when you build this code snippet, it will show you the error: `error C2679: binary '=': no operator found which takes a right-hand operand of type 'int' (or there is no acceptable conversion)`.

0 Reply

Alex Author
Reply to Abhey[21] January 19, 2024 8:47 am

You don't have to use brackets for the last question, but it's a best practice to, so you should.
I have no idea why `x = 5` wouldn't be working.

0 Reply

Hadi
January 16, 2024 12:53 pm

why arent there excercises like write a program that does X ? The explanation is beyond perfect, the only thing that is missing is excercises like w3school. Anyways big thanks to the creators of these !

2 Reply

Long
Reply to Hadi[22] January 23, 2024 7:20 pm

I think compiling the snippets in this section is enough. You can count those at small exercises

0 Reply

uncle_pear
Reply to Hadi[22] January 17, 2024 3:29 am

True

1 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/literals/
3. https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/
4. javascript:void(0)
5. https://www.learncpp.com/cpp-tutorial/introduction-to-expressions/
6. https://www.learncpp.com/
7. https://www.learncpp.com/cpp-tutorial/whitespace-and-basic-formatting/
8. https://www.learncpp.com/introduction-to-literals-and-operators/
9. https://www.learncpp.com/cpp-tutorial/uninitialized-variables-and-undefined-behavior/
10. https://www.learncpp.com/cpp-tutorial/developing-your-first-program/
11. https://gravatar.com/
12. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-595491
13. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-595535
14. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-595521
15. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-595018
16. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-595055
17. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-594224
18. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-593791
19. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-593299
20. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-592845
21. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-592522
22. https://www.learncpp.com/cpp-tutorial/introduction-to-literals-and-operators/#comment-592428